An Improved Hybrid AMPM and ISB Prefetcher

Mark Mansi
mmansi@utexas.edu
Department of Computer Science
University of Texas at Austin

Turing Scholar Honors Thesis Committee:
Calvin Lin (Advisor)
Don Fussell
Etienne Vouga

Abstract

In their recent work, Jain and Lin propose a hybrid prefetcher composed of the Access Map Pattern Matching Prefetcher (AMPM) and the Irregular Stream Buffer (ISB), two state-of-the-art prefetchers [4]. While their hybrid achieves high performance, it can also waste memory bandwidth by making many inaccurate prefetches, which can hurt system performance in bandwidth-constrained environments. We aim to improve their work along two fronts. First, our design improves the accuracy of AMPM. Second, our design identifies whether the current workload is regular or irregular and uses this information to toggle between AMPM and ISB in the hybrid.

While our design does not fully meet these goals, we do show that it improves AMPM accuracy by 7% while decreasing coverage by only 3%. Our solution improves Hybrid accuracy by 4% and bandwidth consumption by 8% on average, though it decreases speedup by 17% compared with the original hybrid.

1 Introduction

Even for today’s simple in-order cores, DRAM latencies are a performance bottleneck for many workloads. The processor can waste hundreds of cycles waiting for a memory request that misses in the caches. One technique for mitigating memory latency is prefetching. Leveraging information about previous memory accesses, prefetchers predict and issue memory requests for addresses before the program needs them. When execution requires these addresses, they will already be on their way from memory, hiding some or all of the memory latency.

Many prefetcher designs have been proposed [3, 4, 6, 7, 8, 10, 12, 13, 14], but generally, they target either regular or irregular streams [4]. Regular streams consist of memory accesses at constant strides from each other (e.g. sequential accesses in an array); irregular streams consist of memory accesses that do not follow constant strides (e.g. following a linked list in memory). Unfortunately, regular prefetchers are typically ineffective at prefetching irregular streams. In some cases, they can even make significant amounts of inaccurate prefetches, polluting the caches, wasting memory bandwidth, and potentially hurting system performance. Meanwhile, irregular prefetchers can waste system resources tracking regular streams that could be more efficiently targeted by regular prefetchers.

It is important for prefetchers to efficiently share memory bandwidth with the processor itself by prefetching accurately. If processor demand memory requests are delayed, execution may be slowed down. Meanwhile, inaccurate prefetching fills the memory subsystem with useless requests that take time and bandwidth to fulfill. Thus, in a bandwidth-constrained environment, inaccurate prefetching can potentially slow down execution.

Our work aims to improve upon the work of Jain and Lin, who combine a regular prefetcher (AMPM [3]) and an irregular prefetcher (their own ISB) into a hybrid prefetcher to achieve higher overall performance. They find that their hybrid can achieve better performance than either AMPM or ISB but also wastes memory bandwidth and energy due to inaccuracy. They note that a hybrid with a more accurate regular prefetcher may be able to match or exceed the performance of their hybrid without wasting as much bandwidth [4].

With the goal of improving upon Jain and Lin’s hybrid, we aim to improve the inaccuracy of AMPM and to find a runtime metric for identifying the regularity or irregularity of a workload. To this end, we make the following findings.

1. We describe access map density, a metric for gauging the irregularity of a workload when running AMPM.

2. We find that workloads are generally composed of distinct phases of regularity and irregularity.

3. We find that when using AMPM, short strides are more useful than long strides.

We use these findings as follows.
First, we make AMPM more accurate at a small cost in coverage by strategically limiting AMPM’s range of strides. We observe that while AMPM checks all strides from -128 to 128, in most cases, only a small range of short strides is useful.

Second, we describe a toggling system between the two component prefetchers of the hybrid. Jain and Lin’s design continually leaves both prefetchers running, potentially causing memory bandwidth contention. We turn off either the regular or irregular prefetcher based on the
irregularity of the workload, as determined by runtime measurements of density.

We use a parallel x86 simulator to evaluate our design. We improve the accuracy of AMPM by 7%, while decreasing its coverage by only 3%. Moreover, our toggling mechanism improves the accuracy of the hybrid by 4% and decreases bandwidth usage by 8% compared to the original hybrid proposed by Jain and Lin. Nonetheless, our toggling mechanism does not fully meet our design goals because it decreases coverage by 11% and speedup by 17% compared with the original hybrid.

This paper is organized as follows. Section 2 describes prior work which forms the context for our work. Section 3 describes our design. In section 4, we evaluate our design. Finally, section 5 presents a summary of our findings and discusses possible directions for future work.

2 Related Work

In this section, we discuss AMPM, ISB, and the hybrid described by Jain and Lin since our work improves on this work. We also give a brief survey of other related work in the field.

2.1 AMPM

The Access Map Pattern Matching Prefetcher (AMPM) is a regular prefetcher originally presented at the Data Prefetching Championship [3]. AMPM has high performance for regular streams, but poor performance for irregular streams [4]. In particular, it can be too aggressive for irregular streams, producing many inaccurate prefetches. This wastes memory bandwidth and energy and pollutes the cache.

Most of AMPM’s on-chip storage requirements go towards maintaining a collection of access maps corresponding to a subset of the memory address space. Each access map is a bitmap in which each “bit” is a 3-state machine representing a given cache line in the memory region the map is tracking. Each cache line starts in the “Init” state. If AMPM prefetches the line, it transitions to the “Prefetched” state. Upon a demand access, the cache line transitions to the “Accessed” state. In this way, AMPM can keep track of the contents of the cache and a history of memory accesses in a compact way. If needed, AMPM may allocate a new access map, possibly evicting an old one to make room [3].

When the prefetcher is triggered, the access map corresponding to the trigger address is loaded. Complex logic is used to check the access maps for strided accesses: for a trigger address \( a \) and stride \( s \), if \( a - s \) and \( a - 2s \) have been accessed, then a stream is detected with stride \( s \), and a prefetch is issued for \( a + s \). This is done for all strides \( s \) from -128 to 128 until a prefetch is issued, starting with short strides [3].

We use AMPM as the regular prefetcher in our hybrid because of its high performance for regular streams. However, we aim to make AMPM more accurate even at a cost in miss coverage. We find that often accuracy can be improved by limiting the range of strides that AMPM uses. This means that less bandwidth is wasted by inaccuracy, potentially improving performance.

2.2 ISB and Hybrid Prefetchers

The Irregular Stream Buffer (ISB) is the irregular prefetcher designed by Jain and Lin that we use in our hy-
The ISB maps consecutive memory accesses to consecutive addresses in an internal structural address space. This linearizes irregular streams by concisely recording repeated address correlations. Then, a regular prefetcher can issue prefetches in the structural address space easily [4].

We do not focus on improving the performance or resource usage of ISB in this work, but we do describe a toggling mechanism for the hybrid that aims to reduce the effects of this high resource usage. We improve upon the hybrid by strategically turning off one of the prefetchers in the hybrid based on the irregularity of the workload. This takes advantage of the strengths of each prefetcher and avoids inaccurate prefetches. During a regular phase of workload, we may save memory, bandwidth, and energy by turning off ISB. During an irregular phase of a workload, we may save bandwidth and energy by stopping AMPM from prefetching.

2.3 Other work
Considerable other work has been done on prefetching. Feedback-Directed Prefetching is another regular prefetcher that uses runtime feedback on its performance to modulate prefetch aggressiveness [14]. Pugsley et al. extend this work with their Sandbox Prefetcher, which tests a collection of stride prefetchers in a sandbox to find the most accurate ones [10]. Either of these regular prefetchers may be used in a hybrid, but we do not explore the idea further in this work.

A couple of designs from the Second Data Prefetching Championship also aim to improve AMPM. Young et al. introduce a design that combines AMPM with the DCPT [2, 15]. Jia et al. focus on improving performance for infrequently-accessed memory and making access maps better reflect cache contents [5]. These designs mainly focus on improving AMPM in a standalone environment, while we aim to improve the behavior of AMPM in the context of a hybrid prefetcher.

3 Our Solution
We describe both an improvement to AMPM and an improvement to the Hybrid prefetcher. To improve AMPM, we propose Range Cutoff. To improve the Hybrid prefetcher, we describe a runtime metric, density, which can be used to identify at runtime the irregularity of a workload. We describe a toggling mechanism that uses density to decide when to turn off the regular or irregular prefetcher.

3.1 Range Cutoff
To improve the accuracy of AMPM we use Range Cutoff. Range Cutoff limits the range of strides available to AMPM. Although AMPM checks strides -128 to 128, we find that most workloads primarily rely on short strides, especially stride 1. Moreover, for irregular workloads, often AMPM has false positives using large strides, identifying coincidentally aligned accesses that are not part of a stream. Range Cutoff instead limits AMPM to using strides $-s$ to $s$, for some positive integer $s$. We find that $s = 4$ seems to work well.

3.2 Access Map Density
Our Hybrid Toggling mechanism uses density to gauge the irregularity of a workload at runtime. The density

![Figure 2: Bandwidth Consumption by Hybrids (in tens of millions of memory operations). Each bar represents a different density threshold. The bar labeled “None” corresponds to the original hybrid. Using a density threshold, we can improve mean bandwidth usage by 8%.

[Image of Figure 2]
Figure 3: Accuracy, coverage, and speedup for Range Cutoff. Mean accuracy increases by 7%, while mean coverage decreases by only 3% and speedup decreases by about 3.5%.
at any point during program execution is the proportion of all access map entries on all allocated access maps that are “Accessed”. In irregular workloads, memory accesses “jump around” memory. This causes AMPM to constantly evict and reallocate access maps, so very few become dense. In contrast, in regular workloads, access maps often become dense and may saturate several times over the course of the program.

Since instantaneous density can vary greatly in a short amount of time, our solution instead measures density over epochs of the workload. The instantaneous density is computed after every prefetch and added to a global counter. This counter is used to compute the average at the end of the epoch. Our design counts the length of an epoch by the number of Last Level Cache (LLC) cache misses; after a certain number of cache misses the current epoch ends and the next one begins. The average density over the previous epoch determines prefetcher behavior in the current one.

An epoch should be long enough to capture coarse-granularity workload behavior but short enough to reflect changes in the workload’s behavior. In our implementation, we use an epoch length of 5000 cache misses because empirically this value seems to work well.

3.3 Hybrid Toggling Mechanism

The goal of our hybrid is to use either AMPM or ISB for any given epoch based on the density. For each epoch, if the previous epoch’s density was lower than a given threshold, we assume the workload is irregular, so prefetches from AMPM are suppressed. Likewise, if the density was at least as high as the threshold, we assume the workload is regular, so prefetches from ISB are suppressed.

It is important to note that in our design, we only suppress prefetches. The prefetchers continue to train on trigger addresses. Our solution may be able to save on storage and energy by turning off one prefetcher completely, but we do not explore the idea further in this work.

4 Evaluation

4.1 Methodology

Simulation Infrastructure. We evaluate our ideas using Sniper Simulator, version 6.1, a parallel x86 simulator. While Sniper does not simulate a full system, it comes within 25% of true hardware performance [1]. The parameters of the simulated system are shown in Figure 4. We believe this configuration is representative of a single core of a modern commercial processor. All prefetchers prefetch into the LLC.

Benchmarks. We test our designs using fourteen memory-intensive benchmarks from SPECint2006 and SPECfp2006 used by Jain and Lin in their evaluation [4] and by Ishii et al. in theirs [3]. All benchmarks are compiled using gcc 4.2 with the -O2 flag and disabling SSE3/4 instructions. All benchmarks are run using the reference inputs.

Using the SimPoint sampling methodology, we generate a set of representative 250 million instruction SimPoints for each benchmark [11]. We then use the PinPlay tool to generate Pinballs for Sniper from the SimPoints [9].

Evaluated Prefetchers. We simulate several configurations: no prefetcher, original AMPM, original Hybrid, AMPM with Range Cutoff, and Hybrid with Density-based Toggling. For each prefetcher, we measure the proportion of prefetches that are accurate, the LLC cache miss coverage, and the performance improvement.

In our simulations, we provision AMPM with 4KB of on-chip storage and ISB with 8KB of on-chip storage and 8MB of off-chip storage. Thus, the Hybrid configurations use a total of 12KB of on-chip storage and 8MB of off-chip storage.

4.2 Results

4.2.1 Range Cutoff

Figures 1 and 3 show the bandwidth consumption, accuracy, coverage, and performance improvement for Range Cutoff. Using the strictest Range Cutoff, [-4,4], the mean accuracy increases by 7%, while the mean coverage decreases by just 3% and mean speedup decreases by about 3.5%. For most benchmarks, the decrease in speedup is small. For sphinx3, the decrease is nearly 10%, and for xalancbmk 6%. omnetpp actually sees an increase in speedup of about 1%, while libquantum, lbm, mcf, astar, and soplex see very little change in speedup.

4.2.2 Hybrid

We evaluate both the original and density-based Hybrids. Figures 2 and 5 show the bandwidth consumption, accuracy, coverage, and performance improvement.
Figure 5: Accuracy, coverage, and speedup for Hybrids. Each bar represents a different density threshold. The bar labeled “None” corresponds to the original hybrid. Using a density threshold, we can improve accuracy by 4%.
The original hybrid outperforms the density-based hybrid. Using a density threshold of 20% yields the best performance among the density-based hybrids. Compared to the original hybrid, using a density threshold of 20% improves accuracy by about 4% and bandwidth consumption by 8%, but decreases coverage by about 11% and speedup by about 17%.

omnetpp, a purely irregular benchmark, and lbm and libquantum, two purely regular benchmarks, do see the expected results. omnetpp has an increase in accuracy of about 20%, the highest of any benchmark. Moreover, none of these benchmarks sees a decrease in coverage or speedup. This highlights the potential inherent in our design for improving bandwidth consumption and accuracy without hurting performance.

5 Conclusions
In this paper, we describe two fronts of potential improvement upon the work of Jain and Lin [4]. First, we describe Range Cutoff, a technique for improving the accuracy of AMPM. Second, we describe density, a metric easily measured at runtime that can be used to guage the irregularity of a workload. We demonstrate a way to use runtime density measurements to potentially decrease the memory bandwidth usage of Jain and Lin’s Hybrid prefetcher. While we do not fully meet our design goals, our results demonstrate potential accuracy and bandwidth usage improvement using our design.

We find that the design spaces for Range Cutoff and the Hybrid are extensive. There were several design choices that we were not able to explore for this work that may improve performance. For example, our Hybrid toggling mechanism always choses either AMPM or ISB, whereas it may be useful at times to use both prefetchers. Also, our design only suppresses prefetches, but it may also be useful to suppress training to save on storage requirements. Yet another consideration is epoch length; we use an epoch length of 5000, but we were not able to fully explore this dimension of the design. Thus, we feel that future work may focus on exploring the rest of this vast design space.

6 Acknowledgments
This project would not have been possible without the help of Akanksha Jain. Her expertise in caching and prefetching and her technical input are much appreciated.

I would also like to thank Hao Wu for his help with simulation infrastructure.

References


